RETDAT-related Thoughts

Time-stamps for event-based requests

Tue, Feb 4, 2003

Event-based time-stamped replies

The current `RETDAT` support allows for access to “time-stamped” data that permits receiving continuous 15 Hz data that can be correlated across multiple front ends. The essence of the scheme is that 7.5 Hz replies include two consecutive 15 Hz cycles of the requested data. Because the data is returned as a structure, it must be processed in a special way by the application in order to achieve this, but at least it is possible. Acnet clients cannot reliably receive 15 Hz data at 15 Hz, because applications operate only roughly at 15 Hz and are asynchronous to the accelerator 15 Hz. To be fair, an attempt to receive 15 Hz replies will more often succeed than fail, but it is not reliable; some 15 Hz replies will be missed, since the application must sample what is in the client system data pool, not a queued reply message.

A possible modification to the basic scheme can permit access to such time-stamped data based not on continuous 15 Hz data but rather on event-based data, such as specifying clock event `0x10` to collect data on all beam cycles, even when they occur in 15 Hz bursts. To see how this can be done, consider the two-word header that precedes the one or two sets of data that the original scheme uses. The first 16-bit word indicates whether one or two sets of data are present in the reply buffer. (Currently, the value of one applies only to the first immediate reply.) The second word specifies the time-stamp—really a 15 Hz cycle number—of the first set of data in the reply buffer. (The time stamp of the second set of data, when one exists, is simply one more than the indicated time-stamp value.) To achieve correlated data across multiple front ends, the time-stamp cycle number is derived from the clock event message that is sent via multicast on the network.

To adapt the scheme to event-based requests, the new logic first needs to allow event-based requests to lead to time-stamped data returns. Replies would be collected whenever the selected event applies to the current 15 Hz cycle. If either of two consecutive cycles has the selected event, then a reply is returned that specifies the appropriate count and the appropriate time-stamp of the first (or only) set. If neither of two consecutive 15 Hz cycles has the selected event, then no reply is returned. Even if the selected clock event occurs continuously at 15 Hz, replies can be returned only at 7.5 Hz. If it occurs at less than 15 Hz, then replies will be returned less often. It appears that the basic scheme can be adapted to deliver only replies on a selected clock event. The new logic required to provide this support is expected to be minimal.

Event-based requests from missing nodes

A server node should be able to complain of missing data from a contributing node in the case of an event-based request. The server node is aware of the clock events, just as are the contributing nodes. This allows the server node to recognize that, on a cycle in which the selected clock event applies, a reply was not received from a particular node and thus to return the tardy status of `36 –7`. This should not be too difficult, because when a reply is received, the current cycle number is captured for the request. If the clock event reply is being built, and if the cycle number last captured for that node is not the current cycle number, then the tardy status is appropriate. In addition, one should be able to send a reminder request to that node, in case it went down and subsequently recovers, so that once it is back up, it will again return replies as a full-fledged contributing node. This same reminder service is already provided for simple periodic requests.

For a case of a node being down when the original request is initialized, there is already support for a missing node in event-based requests via the `36 –8` status that is returned. This addition would merely add the tardy status and reminder features.

Alternate waveform access

There are actually two forms of waveform access supported. One was described earlier that access the waveform buffer, given an `SSDN` that can as well be used to access an ordinary single analog channel reading. From the `CINFO` entry, it can find the location of the waveform buffer, if one exists.

A second type of waveform access can be used to collect data from an IRM 1KHz digitizer, or from the new HRM (10KHz) digitizer. The reply data is comprised of a two-longword header followed by an array of data and time values for each point. This access is not reading out a waveform buffer, but actually continuously samples the data from a circular buffer that the hardware writes. In order to access such data using the usual data request protocol, the reply data is sampled from the hardware circular buffer at the rate that is implied from the number of bytes requested and the reply period.

As an example, say we want to read data at 300 Hz from a 1KHz digitizer that is installed in an IRM. The requester might specify a reply buffer of (8 + 2*20*4) = 168 bytes in length to be returned at 7.5 Hz. This works because at 7.5 Hz, 40 points are accumulated, and each point occupies 4 bytes for the data and time values. The data is in the usual raw data form, and the time is in units of 10 microseconds. The 2-byte time values are actually relative to a 4-byte time value that is the first longword of the header. That 4-byte base time can either be relative to the time the request was initialized, or it can be relative to the occurrence of a selected clock event. As an example, the `SSDN` (for the reading property) would be something like:

```
5202  061F  0112  0010
```

Here, the listype is 82 to specify this kind of reply data, the node number is `0x061F`, the channel number is `0x0112`, and the selected clock event is `0x10`. The times reported in the reply data structure, when the base time is added to the relative times for each point, indicate the time (in 10 μs units) since the last `0x10` clock event.

For the case that a user might want to specify the clock event rather than have it fixed by the `SSDN`, specify a value of `0x0000` in the 4th word, and set the offset flag in the first word, so it would become `0x5212`. Then the offset value specified in the `RETDAT` request is the clock event to be used as a reference for the returned time values. The 16-bit offset value is added to the 32-bit ident value (`0112 0000`) to yield the same channel number but a different clock event.